ELSEVIER

Discrete Applied Mathematics 59 (1995) 153–179

DISCRETE APPLIED MATHEMATICS

A framework for adaptive sorting☆

Ola Petersson[a,*], Alistair Moffat[b]

[a]*Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden*
[b]*Department of Computer Science, The University of Melbourne, Parkville 3052, Australia*

Received 14 July 1992

Abstract

A sorting algorithm is adaptive if it sorts sequences that are close to sorted faster than random sequences, where the distance is determined by some measure of presortedness. Over the years several measures of presortedness have been proposed in the literature, but it has been far from clear how they relate to each other. We show that there exists a natural partial order on the set of measures, which makes it possible to say that some measures are superior to others. We insert all known measures of presortedness into the partial order, and thereby provide a powerful tool for evaluating both measures and adaptive sorting algorithms. We further present a new measure and show that it is a maximal element in the partial order formed by all known measures, and thus that any sorting algorithm that optimally adapts to the new measure also optimally adapts to all other known measures of presortedness. We have not succeeded in developing an optimal algorithm for the new measure; however, we present one that is optimal in terms of comparisons.

Keywords: Sorting; Presortedness; Measures; Partial order

1. Introduction

It is well known that, in a comparison-based model of computation, $\Omega(n \log n)$ time is necessary to sort n elements in both the worst case and the average case [19]. Despite this fact, some instances seem easier than others and can be sorted faster than indicated by the lower bound, for example, an already sorted sequence or the concatenation of two sorted sequences. These should not require the same amount of resources as a randomly permuted sequence. This observation was first made by Burge [2] in 1958, who identified this *instance easiness* with the amount of existing order (*presortedness*) in the sequence, and also proposed measures of existing order.

☆A preliminary version of this paper was presented at the Third Scandinavian Workshop on Algorithm Theory, Helsinki, Finland, July 1992.

*Corresponding author.

SSDI 0166-218X(93)E0160-Z

After being considered by, among others, Cook and Kim [4], Dijkstra [6], Guibas et al. [10], Knuth [12], and Mehlhorn [18, 19], the concept of presortedness was eventually formalized by Mannila [17] in 1985. Mannila studied the question of how the presortedness of a sequence can be measured. Examples of measures of presortedness that he considered are the number of runs and the number of inversions. Mannila also studied the problem of how a sorting algorithm can take advantage of, and thereby adapt to, the existing order. A sorting algorithm is said to be *adaptive* with respect to a measure of presortedness if it sorts all sequences, but performs particularly well on those having a high degree of presortedness according to the measure. The more presorted the input is, the faster it should be sorted. Moreover, the algorithm should adapt without a priori knowing the amount of presortedness. Note that most worst-case optimal sorting algorithms, e.g. Heapsort and Mergesort, do not take existing order within their input into account.

Different measures of presortedness judge different sequences as presorted, and Mannila posed the question of whether one can somehow compare measures. (Comparing their numerical values gives no information, as, for example, the ranges may differ.) In this paper we show that it is indeed possible. We introduce a natural relation, denoted $\supseteq$, which defines a partial order on the set of measures. If M_1 and M_2 are two measures of presortedness, and $M_1 \supseteq M_2$, every sorting algorithm that optimally adapts to M_1 optimally adapts to M_2 as well. Hence, from a sorting algorithmic point of view M_1 is at least as good as M_2. Since we insert all known measures into the partial order, our results have consequences for most of the results obtained earlier in the area of adaptive sorting. More importantly, however, the partial order provides a framework for evaluating both new measures of presortedness and new adaptive sorting algorithms.

We further present a new measure of presortedness that is a maximal element in the known partial order. Thus, every sorting algorithm that optimally adapts to our new measure is optimal with respect to all known measures. The measure is closely related to an algorithm, *Regional Insertion Sort*, that sorts by repeatedly inserting the elements from the input sequence into a *Historical Search Tree* [20]. The algorithm is optimal with respect to the measure if we only count the number of comparisons performed; however, there are sequences for which it uses $\Theta(n \log\log n)$ overhead time that should be sorted in $o(n \log\log n)$ time according to the measure.

The remainder of the paper is organized as follows. In Section 2 we present some well-known measures of presortedness. In Section 3 we recall two simple adaptive sorting algorithms, and define the concept of optimality. In Section 4 we introduce the concept of superiority of a measure to another, which defines the partial order on the set of measures. In Section 5 we extend the partial order to include all known measures of presortedness. In Section 6 we review various adaptive insertion sort algorithms, and, motivated by the weaknesses of these, we design Regional Insertion Sort and define the associated measure of presortedness. In Section 7 we return to the partial order and insert the measures associated with the insertion sorts. Finally, in Section 8 we summarize and comment on the results obtained.

2. Measures of presortedness

We first state some preliminary definitions. Let $X = \langle x_1, \ldots, x_n \rangle$ be a *sequence* of n elements x_i from some totally ordered set. For simplicity, the x_i's are assumed to be distinct. For two sequences, $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$, their *concatenation* XY is the sequence $\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$. A sequence obtained by deleting zero or more elements from X is called a *subsequence* of X. Further, let $|X|$ denote the *length* of X and $\|S\|$ the *cardinality* of a set S. S_n is the set of all permutations of $\{1, \ldots, n\}$ and $\log x = \log_2(\max\{2, x\})$. It is also convenient to extend the asymptotic notation for functions on integers to functions on sequences, as follows. Let f and g be non-negative functions on the set of sequences. We say that $f(X) = O(g(X))$ if there exist constants n_0 and $c > 0$ such that $f(X) \leqslant c \cdot g(X)$, for all X with $|X| > n_0$. Further, $g(X) = \Omega(f(X))$ if $f(X) = O(g(X))$; $f(X) = \Theta(g(X))$ if $f(X) = O(g(X))$ and $f(X) = \Omega(g(X))$; and $f(X) = o(g(X))$ if $f(X) = O(g(X))$ and $g(X) \neq O(f(X))$.

As mentioned before, we evaluate instance easiness in the sorting problem by a *measure of presortedness*, a non-negative integer function on a sequence X that reflects how much X differs from the sorted permutation of X. Note that a measure of presortedness grows with the amount of disorder rather than with the amount of existing order; the "closer" X is to being sorted, the smaller the value of the measure.

Mannila [17] proposed some general conditions that a function should fulfill to qualify as a measure of presortedness. We take a looser approach, however, and only require that the function in some intuitive sense should quantify disorder within the input. One of the conditions that we will adopt is that a measure should evaluate to the same value for two sequences that are order isomorphic. That is, if M is a measure of presortedness and $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ are two sequences in which $x_i < x_j$ if and only if $y_i < y_j$, for all $1 \leqslant i, j \leqslant n$, then we require that $M(X) = M(Y)$. A sequence X of length n can thus be thought of as the element π in S_n which is order isomorphic to X.

To facilitate understanding, we follow Mannila [17] and present some well-known measures of presortedness that can be found in the literature. Not surprisingly, it turns out that the amount of presortedness in a sequence strongly depends on how it is measured. For now, it is left to the reader to find sequences that are intuitively almost in sorted order but for which the measures are asymptotically maximized.

The most common measure is probably the number of *inversions* in a sequence, that is, the number of pairs of elements that are in wrong order:

$$Inv(X) = \|\{(i, j) \mid 1 \leqslant i < j \leqslant n \text{ and } x_i > x_j\}\|.$$

Like most measures of presortedness, *Inv* has an operational interpretation, in the sense of the minimum number of operations of a certain kind needed to bring a sequence into sorted order. For *Inv*, this operation is the exchange between two adjacent elements. Note that *Inv* tells the exact number of exchanges made by Straight Insertion Sort [12].

Another well-known measure of presortedness is the number of *runs* within the input. By a run we mean an ascending consecutive subsequence of maximum length:

$$Runs(X) = \|\{i \mid 1 \leqslant i < n \text{ and } x_i > x_{i+1}\}\| + 1.$$

Runs does not admit an as natural operational interpretation as did the number of inversions; however, if there are few runs, the sequence can be sorted quickly by repeated pairwise merging.

The third measure considered by Mannila was *Rem*, the minimum number of elements that need be removed to leave a sorted sequence:

$$Rem(X) = n - \max\{k \mid X \text{ has an ascending subsequence of length } k\}.$$

Note that, in contrast to the definition of *Runs*, the subsequence does not have to be consecutive in X. From an operational point of view, *Rem* tells the minimum number of element moves needed to produce the sorted output.

The fourth well-known measure we would like to present is operational by definition, namely

$$Exc(X) = \text{the minimum number of exchanges of arbitrary elements required to bring } X \text{ into sorted order.}$$

3. Adaptive sorting algorithms

As stated before, an adaptive sorting algorithm is a sorting algorithm that adapts to the presortedness within the input, measured in some way. The more presorted a sequence is, the faster it should be sorted. Moreover, the algorithm should adapt without any a priori knowledge of the amount of presortedness. The algorithm may, if it wishes, *calculate* the amount of presortedness, but any time spent doing so is counted as part of the running time.

Besides Straight Insertion Sort, which is not worst-case optimal, the most widely known adaptive sorting algorithm is Natural Mergesort [12]. Presented with a sequence X, this algorithm starts by finding the runs by a linear time scan. These runs are then merged pairwise, resulting in about half as many longer runs. Pairwise merging of the new runs is then continued until there is just one run left. It is easy to see that the time consumed by Natural Mergesort on a sequence X of length n is $\Theta(n \log Runs(X))$.

Another simple adaptive sorting algorithm is by Cook and Kim [4]. The algorithm, which is adaptive with respect to *Rem*, consists of three phases. First, linear time is spent to remove $\Theta(Rem(X))$ elements from the input sequence X such that what remains in X is an ascending subsequence of X. Second, the removed elements are sorted separately. Third, the sorted sequence formed by the removed elements is merged with the sorted sequence that remained in X. If a worst-case optimal

algorithm is applied in the second phase, the total time consumed by the algorithm on a sequence X of length n is $\Theta(n + Rem(X) \log Rem(X))$. For details, see e.g. [15].

We next turn to the question of whether one can sort even faster with respect to *Runs* and *Rem*. That is, are the given algorithms "optimally" adaptive, or is it possible to take more advantage of the presortedness measured by these measures than we have managed to do?

The concept of an optimal algorithm with respect to a measure of presortedness was given in a general form by Mannila [17]. We use the following equivalent definitions.

Definition 1. Let M be a measure of presortedness. Then, for any $k \geqslant 0$ and $n \geqslant 1$,

$$below_M(n, k) = \{\pi \mid \pi \in S_n \text{ and } M(\pi) \leqslant k\}.$$

The set $below_M(n, k)$ contains all permutations that, according to M, are at least as presorted as the sequences X with $M(X) = k$.

Next, we introduce a notation for the number of comparisons needed to sort the permutations in a *below*-set:

Definition 2. Let M be a measure of presortedness, and T_n the set of binary comparison trees for the set S_n. Then, for any $k \geqslant 0$ and $n \geqslant 1$,

$$C_M(n, k) = \min_{T \in T_n} \max_{\pi \in below_M(n,k)} \{\text{depth of } \pi \text{ in } T\}.$$

Finally, we define the notion of optimality of an adaptive sorting algorithm.

Definition 3. Let M be a measure of presortedness, and A a comparison-based sorting algorithm that uses $T_A(X)$ steps on input X. We say that A is *M-optimal*, or *optimal with respect to* M, if $T_A(X) = O(C_M(|X|, M(X)))$.

The following theorem is helpful when proving optimality of an algorithm.

Theorem 1. *Let M be a measure of presortedness. Then*

$$C_M(n, k) = \Theta(n + \log \| below_M(n, k) \|).$$

Proof. Since any binary comparison tree for the set $below_M(n, k)$ has at least $\| below_M(n, k) \|$ leaves, its height is at least $\log \| below_M(n, k) \|$. Hence, we have $C_M(n, k) \geqslant \log \| below_M(n, k) \|$. Moreover, any comparison tree in T_n must examine each element at least once. Consequently, its height must be at least linear in n. We can now conclude that $C_M(n, k) = \Omega(n + \log \| below_M(n, k) \|)$.

The upper bound follows from a result by Fredman [9], who proved that for any set $S \subseteq S_n$, there exists a comparison tree of height at most $2n + \log \|S\|$ that differentiates between the elements of S. □

Remark. Mannila [16] proved that despite the tightness of the bound in Theorem 1 there are measures for which there is no optimal algorithm. That is, even if there exists an optimal algorithm for each particular value of n and k, there is no algorithm that achieves the bound for all possible values.

According to Theorem 1 a lower bound for comparison-based sorting algorithms with respect to a measure of presortedness is obtained by bounding the cardinality of the *below*-set from below. For demonstrational purposes, let us derive a lower bound on $C_{Runs}(n, k)$. (This was also done by Mannila [17]; though his proof is somewhat longer.)

Lemma 2. $C_{Runs}(n, k) = \Omega(n \log k)$.

Proof. To derive a lower bound on the cardinality of $below_{Runs}(n, k)$, we partition $\{1, \ldots, n\}$ into k subsets $S_1, \ldots, S_k$ each of size n/k. Let π be the permutation corresponding to the concatenation of the *sorted* sets, taken in order. Then, $Runs(\pi) \leqslant k$, and so $\pi \in below_{Runs}(n, k)$. Counting the number of different ways of performing the initial partitioning gives

$$\|below_{Runs}(n, k)\| \geqslant n!/\left(\frac{n}{k}!\right)^k,$$

which after taking the logarithm and applying Theorem 1 completes the proof. □

Above, we showed that Natural Mergesort matches the bound stated in Lemma 2, and thus, by Definition 3, we can conclude that Natural Mergesort is optimal with respect to *Runs*. For *Rem*, it has been shown that $C_{Rem}(n, k) = \Omega(n + k \log k)$ [17]. Hence, Cook and Kim's algorithm is *Rem*-optimal.

In general, proving that a sorting algorithm is optimal with respect to a measure of presortedness is done in two parts. One part is a lower bound on $C_M(n, k)$, which is obtained by bounding the cardinality of the *below*-set from below, and often involves a combinatorial construction. The other part is an upper bound on the time consumed by an algorithm that adapts to M, expressed in terms of the measure.

4. A partial order on the measures

Presented with a new measure of presortedness, a natural question to ask is whether it is somehow related to some other measures. Is it even "superior" to other measures, whatever that means?

Let us examine the relationship between the measures *Runs* and *Rem*. By the definition of *Runs*, each run in a sequence X, except the last one, is defined by a consecutive pair of elements x_i and x_{i+1}, for which it holds that $x_i > x_{i+1}$. At most

one of these elements can belong to a longest ascending subsequence in X. Consequently, at least one of them contributes to $Rem(X)$. It follows that $Runs(X) \leqslant Rem(X) + 1$, for any sequence X.

At first, it is tempting to conclude that *Runs* is therefore 'at least as good' a measure as *Rem*, because any sequence with a low *Rem* value has also a low *Runs* value. However, we need to be very careful. Since we are concerned with sorting, what we would really like to prove is that any sequence can be sorted at least as fast by a *Runs*-optimal algorithm as by a *Rem*-optimal algorithm, and this is not true. Consider the sequence

$$X = \langle 1, 2, 3, \ldots, n - \sqrt{n}, n, n - 1, n - 2, \ldots, n - \sqrt{n} + 1 \rangle$$

for which $Runs(X) = \sqrt{n}$ and $Rem(X) = \sqrt{n} - 1$. As $C_{Runs}(n, \sqrt{n}) = \Theta(n \log n)$, a *Runs*-optimal sorting algorithm may spend $\Theta(n \log n)$ time on X, while any *Rem*-optimal sorting algorithm, e.g., Cook and Kim's, must complete the sorting in $O(C_{Rem}(n, \sqrt{n} - 1)) = O(n)$ time.

Motivated by the preceding discussion, we make the following definition.

Definition 4. Let M_1 and M_2 be measures of presortedness. We say that

- M_1 is *superior* to M_2, denoted $M_1 \supseteq M_2$, if

$$C_{M_1}(|X|, M_1(X)) = O(C_{M_2}(|X|, M_2(X)));$$

- M_1 is *strictly superior* to M_2, denoted $M_1 \supset M_2$, if $M_1 \supseteq M_2$ and $M_2 \not\supseteq M_1$;
- M_1 and M_2 are *equivalent*, denoted $M_1 \equiv M_2$, if $M_1 \supseteq M_2$ and $M_2 \supseteq M_1$;
- M_1 and M_2 are *independent* if $M_1 \not\supseteq M_2$ and $M_2 \not\supseteq M_1$.

The importance of Definition 4 becomes evident when combined with Definition 3.

Theorem 3. *Let M_1 and M_2 be measures of presortedness.*

- *If $M_1 \supseteq M_2$ then every M_1-optimal sorting algorithm is M_2-optimal.*
- *If $M_1 \supseteq M_2$ and A is a sorting algorithm that is not M_2-optimal then A is not M_1-optimal.*

In order to show that a measure M_1 is superior to another measure M_2 we need an upper bound on $C_{M_1}(n, k)$, which can be obtained from an upper bound on an algorithm that adapts to M_1.

Lemma 4. *Let M be a measure of presortedness, for which $C_M(n, k) = \Omega(f(n, k))$, where f is non-decreasing on k. If there exists a comparison-based sorting algorithm A that sorts a sequence X in $C_A(X) = O(f(|X|, M(X)))$ comparisons, then $C_M(n, k) = \Theta(f(n, k))$.*

Proof. Since A defines a comparison tree in T_n, the definition of $C_M(n, k)$ gives

$$C_M(n,k) \leqslant \max_{\pi \in below_M(n,k)} \{C_A(\pi)\} = O\left(\max_{0 \leqslant j \leqslant k} \{f(n,j)\} \right) = O(f(n,k)),$$

since f is nondecreasing on j. □

Remark. The restriction that $f(n, k)$ is non-decreasing is necessary for some measures. For instance, for *Inv*, one can certainly prove that

$$C_{Inv}(n,k) = \Omega\left(n \log\left(\frac{\min\{k, \binom{n}{2} - k\}}{n} \right) \right),$$

and there exists an algorithm that matches this bound, namely Adaptive Heapsort [13]. However, Guibas et al. [10] proved that $C_{Inv}(n, k) = \Theta(n \log(k/n))$.

Using Lemma 4 and the lower bounds on $C_M(n, k)$ mentioned in the previous section it follows that $C_{Runs}(n, k) = \Theta(n \log k)$ by the upper bound on Natural Mergesort, and that $C_{Rem}(n, k) = \Theta(n + k \log k)$ by the upper bound on Cook and Kim's algorithm.

Consider now the measures *Rem* and *Exc*. Carlsson et al. [3] proved that $C_{Exc}(n, k) = \Omega(n + k \log k)$ and $Rem(X) \leqslant 2Exc(X)$, for any sequence X.

Lemma 5. *Rem* $\supseteq$ *Exc*.

Proof. From the above given upper bound on $C_{Rem}(n, k)$ and the results of Carlsson et al. we have

$$\begin{aligned} C_{Rem}(|X|, Rem(X)) &= \Theta(|X| + Rem(X) \log Rem(X)) \\ &= O(|X| + Exc(X) \log Exc(X)) \\ &= O(C_{Exc}(|X|, Exc(X))), \end{aligned}$$

from which the claim follows by Definition 4. □

To prove that *Rem* is *strictly* superior to *Exc* we need the following lemma.

Lemma 6. *Let M_1 and M_2 be measures of presortedness. If, for all constants n_0 and $c > 0$, there exists a sequence X of length $n > n_0$ such that $C_{M_1}(n, M_1(X)) < c \cdot C_{M_2}(n, M_2(X))$, then $M_2 \not\supseteq M_1$.*

Theorem 7. *Rem* $\supset$ *Exc*.

Proof. By Definition 4 and Lemma 5 it remains to prove *Exc* $\not\supseteq$ *Rem*. Consider the sequence $X = \langle n, 1, 2, \ldots, n-1 \rangle$. We have $Rem(X) = 1$ and $Exc(X) = n - 1$. Since $C_{Rem}(n, 1) = \Theta(n)$ and $C_{Exc}(n, n-1) = \Omega(n \log n)$, Lemma 6 gives *Exc* $\not\supseteq$ *Rem*. □

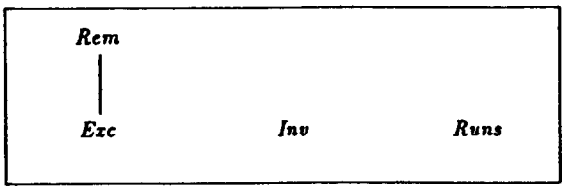

Fig. 1. The partial order $\mathcal{P}(\mathcal{M}, \supseteq)$ obtained so far.

In general, proving that a measure M_1 is superior to another measure M_2 involves the steps we went through for *Rem* and *Exc* above. First, we need an upper bound on $C_{M_1}(n, k)$, which is given by the number of comparisons spent by a sorting algorithm that adapts to M_1. Second, we need a lower bound on $C_{M_2}(n, k)$. The upper bound is then shown to match the lower bound by deriving an upper bound on M_1 in terms of M_2. To prove that M_1 is strictly superior to M_2, we further give a sequence X of length n, which can be extended to arbitrary large n, for which $C_{M_2}(n, M_2(X))$ is asymptotically greater than $C_{M_1}(n, M_1(X))$.

Let us return to the comparison of *Runs* and *Rem*. We gave a sequence proving that $Runs \not\supseteq Rem$. To see that the two measures are independent consider the sequence

$$X = \left\langle \frac{n}{2} + 1, \frac{n}{2} + 2, \ldots, n, 1, 2, \ldots, \frac{n}{2} \right\rangle.$$

Here $Runs(X) = 2$ and $Rem(X) = n/2$. Now, since $C_{Runs}(n, 2) = \Theta(n)$ and $C_{Rem}(n, n/2) = \Omega(n \log n)$, it follows that $Rem \not\supseteq Runs$, by Lemma 6. Hence, by Definition 4, we can conclude that *Runs* and *Rem* are independent.

Let $\mathcal{M}$ be the set of all measures of presortedness. The relation $\supseteq$ defines a partial order on $\mathcal{M}$, which we will denote by $\mathcal{P}(\mathcal{M}, \supseteq)$, and illustrate by a Hasse diagram; see Fig. 1.

The transitivity of $\supseteq$ is a very helpful property when comparing measures of presortedness. For instance, as $Rem \supseteq Exc$ and $Rem \not\supseteq Runs$, it follows that $Exc \not\supseteq Runs$. In the sequel we will see that among the measures we have seen so far, *Rem* and *Exc* are the only ones that are related with respect to $\supseteq$. Hence, no arcs are missing in Fig. 1.

5. Extending the partial order

We insert all measures of presortedness that appear in the literature into the partial order $\mathcal{P}(\mathcal{M}, \supseteq)$. Definitions of the measures, their ranges, and tight bounds on $C_M(n, k)$ are listed in Table 1.

The relations between the measures in Table 1 have been investigated in several papers; indeed, the introduction of measures such as *Osc* and *Block* was motivated by their superiority to known measures. Fig. 2 shows the partial order that results from inserting all of the measures in Table 1.

Table 1
All measures of presortedness that appear in the literature. Here, π is the permutation of $\{1, 2, \ldots, n\}$ for which $\pi(i) = rank(x_i, X) + 1$, for $1 \leqslant i \leqslant n$, where $rank(x_i, X) = \|\{x_j \mid 1 \leqslant j \leqslant n \text{ and } x_i > x_j\}\|$. For each measure, the first reference refers to the paper in which the measure is used, or suggested to be used, for quantifying presortedness

Measure	Range	$C_M(n, k)$	References
$m_0(X) = 0$	0	$\Theta(n \log n)$	[17]
$m_{01}(X) = 0$ if X is sorted, and 1 otherwise	0, 1	$\Theta(n + kn \log n)$	[17]
$Max(X) = \max_{1 \leqslant i \leqslant n} \lvert i - \pi(i) \rvert$	$0 \ldots n-1$	$\Theta(n \log k)$	[8, 3]
$Par(X) = \max\{j - i \mid 1 \leqslant i \leqslant j \leqslant n \text{ and } x_i \geqslant x_j\}$	$0 \ldots n-1$	$\Theta(n \log k)$	[8]
$Inv(X) = \|\{(i,j) \mid 1 \leqslant i < j \leqslant n \text{ and } x_i > x_j\}\|$	$0 \ldots \binom{n}{2}$	$\Theta(n \log(k/n))$	[12, 10, 18, 17]
$DS(X) = \sum_{i=1}^{n} \lvert i - \pi(i) \rvert$	$0 \ldots \lfloor n^2/2 \rfloor$	$\Theta(n \log(k/n))$	[4, 7, 5, 21]
$Osc(X) = \sum_{i=2}^{n} \|\{j \mid 1 \leqslant j \leqslant n$ and $\min\{x_{i-1}, x_i\} < x_j < \max\{x_{i-1}, x_i\}\}\|$	$0 \ldots \lfloor n(n-2)/2 \rfloor$	$\Theta(n \log(k/n))$	[13]
$Runs(X) = \|\{i \mid 1 \leqslant i < n \text{ and } x_i > x_{i+1}\}\| + 1$	$1 \ldots n$	$\Theta(n \log k)$	[12, 17]
$SUS(X) = \min\{k \mid X \text{ is a shuffle of } k \text{ upsequences}\}$	$1 \ldots n$	$\Theta(n \log k)$	[14, 1]
$Enc(X) =$ the number of encroaching sequences	$1 \ldots \lceil n/2 \rceil$	$\Theta(n \log k)$	[22, 14]
$SMS(X) = \min\{k \mid X \text{ is a shuffle of } k \text{ monotone sequences}\}$	$1 \ldots \lfloor \sqrt{2n + 1/4} - 1/2 \rfloor$	$\Theta(n \log k)$	[14, 1]
$Exc(X) =$ the minimum number of pairwise exchanges required to sort X	$0 \ldots n-1$	$\Theta(n + k \log k)$	[17, 3]
$Ham(X) = \|\{i \mid 1 \leqslant i \leqslant n \text{ and } i \neq \pi(i)\}\|$	$0 \ldots n$	$\Theta(n + k \log k)$	[7, 8, 21]
$Rem(X) = n - \max\{k \mid X \text{ has an ascending subsequence of length } k\}$	$0 \ldots n-1$	$\Theta(n + k \log k)$	[4, 17]
$Block(X) = \|\{i \mid 1 \leqslant i < n \text{ and } \pi(i) + 1 \neq \pi(i+1)\}\| + 1$	$1 \ldots n$	$\Theta(n + k \log k)$	[3]

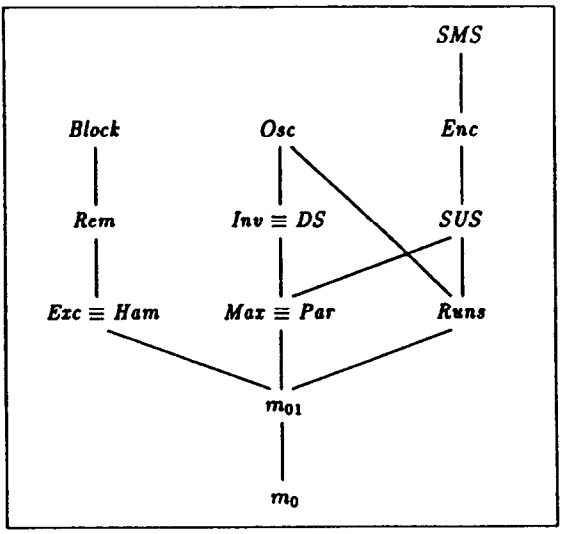

Fig. 2. The partial order for all measures in Table 1.

Although motivated less formally, most of the relations in Fig. 2 can be concluded from results appearing in papers published elsewhere, and we refer the reader to those papers rather than repeating the proofs here.

- The chain $Block \supset Rem \supset Exc$ was proved by Carlsson et al. [3].
- The chain $Osc \supset Inv \supset Max$ and $Osc \supset Runs$ was proved by Levcopoulos and Petersson [13].
- The chain $SMS \supset Enc \supset SUS \supset Runs$ and $SUS \supset Max$ was proved by Levcopoulos and Petersson [14].
- The relations $Exc \supset m_{01}$, $Max \supset m_{01}$, $Runs \supset m_{01}$, and $m_{01} \supset m_0$ are obvious.

The only relations in Fig. 2 that have not been verified are the three equivalences. Let us start by showing that the Hamming distance, *Ham*, is equivalent to *Exc*.

Theorem 8. $Exc \equiv Ham$.

Proof. We first prove that, for any sequence X,

$$Exc(X) + 1 \leqslant Ham(X) \leqslant 2Exc(X). \tag{1}$$

We start with the left-hand inequality. Let X be a sequence with $Ham(X) = k$. Consider the subsequence of X that contributes to $Ham(X)$. X can be sorted by permuting only the elements in this subsequence. Since it contains k elements, the permuting can be done using at most $k - 1$ exchanges, and thus $Exc(X) \leqslant k - 1$.

To prove the right-hand inequality, consider a sequence of $Exc(X)$ exchanges that brings the sorted permutation of a sequence X into X. Each exchange performed can

remove at most two elements from their correct positions. We conclude that $Ham(X) \leqslant 2Exc(X)$.

Combining Eq. (1) with the bounds on $C_{Exc}(n, k)$ and $C_{Ham}(n, k)$, given in Table 1, yields

$$\begin{aligned} C_{Exc}(|X|, Exc(X)) &= \Theta(|X| + Exc(X)\log Exc(X)) \\ &= \Theta(|X| + Ham(X)\log Ham(X)) \\ &= \Theta(C_{Ham}(|X|, Ham(X))), \end{aligned}$$

which completes the proof, by Definition 4. □

The remaining equivalences in Fig. 2 can be proved similarly.

- The equivalence $Max \equiv Par$ follows easily from the fact that

  $$Max(X) \leqslant Par(X) < 2Max(X),$$

  as proved by Estivill-Castro and Wood [8], in combination with the bounds on $C_M(n, k)$ for the two measures.
- The equivalence $Inv \equiv DS$ follows easily from the fact that

  $$Inv(X) \leqslant DS(X) \leqslant 2Inv(X),$$

  as proved by Diaconis and Graham [5], in combination with the bounds on $C_M(n, k)$ for the two measures.

Up to now we have argued for all arcs in Fig. 2. In order to show that no arcs are missing, i.e., that it shows all relations among the measures, we must show a number of independence results between the measures. For convenience, the proofs of these results are postponed till Section 7.

It is natural to ask whether there is a single measure, superior to all measures appearing in Fig. 2. Indeed such a measure exists, namely

$$M(X) = \min\{C_{Block}(|X|, Block(X)), C_{Osc}(|X|, Osc(X)), C_{SMS}(|X|, SMS(X))\}.$$

However, this measure is somewhat artificial. We rather seek a measure with the same properties, but which combines the involved measures in a natural fashion. Starting from a general insertion sort algorithm, the next section is devoted to finding such a measure.

6. Insertion sort algorithms

The following describes a generic insertion sort algorithm [17].

procedure *Insertion Sort* (*X*: *sequence*)
  for $i := 1$ **to** n **do**
    Insert x_i into the sorted sequence formed by $x_1, \ldots, x_{i-1}$
end

The efficiency and adaptivity of the algorithm depends on how the sorted sequence is implemented. Using an array gives Straight Insertion Sort, running in $\Theta(n + Inv(X))$ time, which might be as large as $\Theta(n^2)$.

6.1. A-Sort

Mehlhorn [19] suggested the use of an augmented 2–3-tree, in which an insertion can be made in $\Theta(\log l_i)$ amortized time, where l_i is the distance from the end of the sequence to the position where x_i is inserted. (The search for the insertion position takes $O(\log l_i)$ time in the worst case, and the actual insertion takes $O(1)$ amortized time.) That is, it supports exponential and binary search starting from the end of the sequence. The total time consumed by the algorithm, called A-Sort, is $O(n \log(Inv(X)/n))$, which is *Inv*-optimal.

6.2. Local Insertion Sort

Mannila [17] proposed that instead of being oblivious and start searching from the last element in the sequence, one should exploit locality, and start from the position at which the previous insertion took place. To implement the algorithm, called Local Insertion Sort, he stored the sorted sequence in a finger (search) tree [19], letting the finger point at the most recently inserted element. In this way an element x_i is inserted in $O(\log d_i)$ amortized time, where

$$d_i = \|\{j \mid 1 \leqslant j < i \text{ and } \min\{x_{i-1}, x_i\} < x_j < \max\{x_{i-1}, x_i\}\}\| + 1,$$

that is, the distance from the previous insertion point to the sought position. Thus, Local Insertion Sort runs in time $T_{LIS}(X) = \Theta(\sum_{i=2}^{n} \log d_i)$. Let us consider the measure of presortedness obtained by multiplying the d_i's:

$$Loc(X) = \prod_{i=2}^{n} d_i.$$

(An equivalent measure, $logDist(X) = \sum \log d_i$, was studied by Katajainen et al. [11].) Then $T_{LIS}(X) = O(n + \log Loc(X))$.

Lemma 9. $C_{Loc}(n, k) = \Omega(n + \log k)$.

Proof. We bound the size of $below_{Loc}(n, k)$ from below. Divide the identity permutation in S_n into $n/k^{1/n}$ parts of equal length $k^{1/n}$, and permute the elements within each part. Then all elements are in their correct part, and thus, $d_i < k^{1/n}$. Hence, for every permutation π, constructed in this way, we have $Loc(\pi) < \prod k^{1/n} \leqslant k$, that is, $\pi \in below_{Loc}(n, k)$. Counting the number of different π's constructed this way gives

$$\|below_{Loc}(n, k)\| \geqslant (k^{1/n}!)^{n/k^{1/n}}.$$

Taking the logarithm and applying Theorem 1 completes the proof. □

Since Local Insertion Sort was shown to match the bound stated in the above lemma, we have, by Definition 2, the following theorem.

Theorem 10. *Local Insertion Sort spends* $T_{LIS}(X) = O(n + \log Loc(X))$ *time to sort any sequence* X *of length* n, *which is Loc-optimal.*

Furthermore, by Lemma 4, we have the following corollary.

Corollary 11. $C_{Loc}(n, k) = \Theta(n + \log k)$.

Apart from being a natural measure for quantifying spatial locality in a sequence to be sorted, *Loc* is important in that it is superior to several independent measures, which will be shown in Section 7.

6.3. Historical Insertion Sort

Local Insertion Sort is based on the assumption that if the input is presorted, most insertions will occur close to the previous insertion, where the closeness is determined by the distance in *space*. Another natural way of measuring the distance is in terms of *time*. That is, we might consider the number of insertions performed since one of the elements adjacent to the insertion position was inserted. To make a formal definition of the measure, we first extend the definition of d_i:

$$d_{i,j} = \|\{k \mid 1 \leqslant k < i \text{ and } \min\{x_i, x_j\} < x_k < \max\{x_i, x_j\}\}\| + 1.$$

Hence, $d_{i,j}$ tells the distance from x_j, $j < i$, to the insertion position of x_i. Note that $d_i = d_{i,i-1}$. Now, for $i > 1$ let

$$t_i = \min\{j \mid 1 \leqslant j < i \text{ and } d_{i,i-j} = 1\}.$$

As t_i tells the amount of history needed for inserting x_i, we call the corresponding measure of presortedness *Hist*, and define it analogously to *Loc*, that is,

$$Hist(X) = \prod_{i=2}^{n} t_i.$$

By analogy with Local Insertion Sort we can define a *Historical Insertion Sort* that performs exponential and binary search in time rather than in space. A data structure that supports the insertions required is the *Historical Search Tree* [20]. Essentially, this dictionary data structure is an ordered forest of $\Theta(\log \log n)$ finger trees, where the jth tree $T_j, j \geqslant 1$, contains the $2^{2^{j-1}} - 1$ most recently inserted elements. Nodes representing the same dictionary element in 'adjacent' trees are threaded, that is, a node in tree T_j representing element x is threaded with the node representing x in tree T_{j+1}. The structure can be thought of as a finger tree storing all elements on top of which is a hierarchy of finger trees that ensure that the more recently an element was inserted the higher a node representing it is located.

To search for an element x_i in a Historical Search Tree we search the trees $T_1, T_2, \ldots$ until a node representing x_i is found. When inserting an element x_i we rather search for the gap where x_i is to be inserted. The gap is found once we have located either of the two elements that define it in the bottom tree T_K. In order to decide if this has happened we let the threads 'wrap around' so that during the search downwards in the hierarchy, when a nearest neighbor of x_i in one tree has been found, we can follow the threads to check whether the element found is a nearest neighbor in T_K as well. By the definition of the data structure, the 'youngest' of x_i's nearest neighbors is located in tree T_f, where $f = O(\log\log t_i)$, and the cost of searching trees $T_1, T_2, \ldots, T_f$ is logarithmic in the size of each tree, and thus totals

$$\sum_{j=1}^{f} O(2^j) = O(2^f) = O(\log t_i).$$

Once the gap has been found it remains to update the data structure to reflect that x_i is now the most recently inserted element. This involves inserting x_i into all trees and deleting the 'oldest' element from each tree. The former task is easily done by inserting x_i immediately adjacent to x_i's nearest neighbor in $T_f, T_{f+1}, \ldots, T_K$, following the thread from the node found in T_f. Insertions into trees $T_1, T_2, \ldots, T_{f-1}$ can be taken care of when these trees are searched on the way down in the hierarchy. Hence, no further searches, and therefore no comparisons, are required during the insertions. The amortized time taken is O(1) per tree, which totals $O(\log\log n)$. If the size of the bottom tree T_K exceeds $2^{2^{K-1}} - 1$ due to the insertion, it is duplicated and the threads are extended to the copy. In this way, the hierarchy grows by one level, and an amortization argument shows that it takes only constant amortized time per insertion. We must also delete the oldest element from each tree. In order to do this efficiently a move-to-front list is maintained, which keeps track of the 'victims' and which is updated after each operation. Using the move-to-front list the deletions can be performed within the same resource bounds as the preceding insertions. For further details, we refer the reader to the paper that introduces the Historical Search Tree [20].

Theorem 12. *Historical Insertion Sort spends* $C_{HIS}(X) = O(n + \log Hist(X))$ *comparisons and* $T_{HIS}(X) = O(n \log\log n + \log Hist(X))$ *time to sort any sequence* X *of length* n.

Proof. By the above discussion, the number of comparisons performed is given by

$$O\left(\sum_{i=2}^{n} \log t_i\right) = O\left(n + \log \prod_{i=2}^{n} t_i\right) = O(n + \log Hist(X)).$$

The time taken is linear in the number of comparisons plus $O(\log\log n)$ per insertion. The bound follows. □

Replacing d_i by t_i in the proof of Lemma 9 gives that $C_{Hist}(n, k) = \Omega(n + \log k)$. Hence, by Lemma 4 and Theorem 12, we have the following result.

Corollary 13. $C_{Hist}(n, k) = \Theta(n + \log k)$.

In Section 7 we investigate how *Hist* is related to other measures of presortedness.

6.4. *Regional Insertion Sort*

There exist sequences that are intuitively almost sorted but for which most insertions are costly regardless of whether the cost is a function of distance in time or of distance in space. Katajainen et al. [11] discussed a variant of Local Insertion Sort that exploits locality in time as well as in space, to some extent. The idea was simply to finger the p most recently inserted elements, allowing more than just the most recent one to influence the insertions. The fingers were stored in a search tree, and the algorithm can be viewed as first searching backwards in time to refine the target interval in the finger tree, and then searching the interval in space, starting from both endpoints simultaneously. They also provided a technique to dynamize the algorithm, such that an appropriate number of fingers is allocated during the sorting process. The resulting algorithm is never asymptotically slower than Local Insertion Sort, and significantly faster on infinitely many sequences. However, a major drawback is that the temporal search is not sufficiently adaptive; it is just binary, while the search in space is exponential and binary. Therefore, each insertion takes $\Omega(\log p)$ comparisons.

In order to improve upon the above variants of Insertion Sort, let us make an analogy in which their weaknesses and strengths become even more apparent. In the following, 'remembering' and 'watching' correspond to searching in history and space, respectively. Then, Straight Insertion Sort does not remember anything and is blind. A-sort does not remember anything either, but it has good eyes. Local Insertion Sort has good eyes and one unit of memory, but this is not good enough. Historical Insertion Sort remembers everything but is blind. Finally, the multiple finger algorithm has a reasonably good memory and good eyes; however, it is unable to see till it has finished remembering. It is evident that a better algorithm should have both a good memory and good eyes, as well as being capable of combining these qualities efficiently.

This leads us to *Regional Insertion Sort*, where the assumption made is that most insertions will occur close to *some* element which was inserted recently, but neither necessarily close to the most recent one, nor immediately adjacent to a recent element. Hence, the new algorithm is more generous than its ancestors in what sequences are regarded as presorted.

Before presenting the algorithm, we describe an associated measure of presortedness. When inserting x_i, suppose we first traverse a distance t in time to find an element from which to start searching in space. Then the shortest possible total distance covered is given by

$$r_i = \min_{1 \leqslant t < i} \{t + d_{i,i-t} - 1\},$$

where t and $d_{i,i-t}$ correspond to the time and space components, respectively. Note that choosing $t = 1$ gives $r_i \leqslant d_{i,i-1} = d_i$, and choosing $d_{i,i-t} = 1$ gives $r_i \leqslant t_i$. Hence, r_i combines the distances in time and space, and can be viewed as two-dimensional. Next, we define the measure *Reg* as the product of the r_i's:

$$Reg(X) = \prod_{i=2}^{n} r_i.$$

Replacing d_i by r_i in Lemma 9 gives $C_{Reg}(n, k) = \Omega(n + \log k)$, which is the bound that Regional Insertion Sort is to match.

To adapt to *Reg*, Regional Insertion Sort searches for the insertion position by an exponential and binary search in history and space simultaneously, interleaving the operations. To support the algorithm we once again apply the Historical Search Tree, using a slightly modified insertion algorithm. The details of the algorithm, which is sketched below, can be found in [20].

Suppose that x_i is the next item to be inserted. The searching strategy already described for Historical Insertion Sort pauses after each tree T_j and performs a 'proximity check' in the bottom tree T_K to see whether or not the gap containing x_i has been located. This check consists of examining the T_K successor of the T_j predecessor of x_i, and the T_K predecessor of the T_j successor of x_i. These two checks require $O(j)$ time, following the threads upwards until they wrap around.

In fact, we can afford to spend more time in this checking phase, since $O(2^j)$ time has just been spent searching in tree T_j. Suppose we allow the proximity check in T_K to also spend $\Theta(2^j)$ time trying to locate the insertion point for x_i. Since T_K is a finger tree, it is possible in $\Theta(2^j)$ time to search a distance (in space) of 2^{2^j} elements. The searching phase stops whenever x_i's insertion point is found in T_K to lie within 2^{2^j} elements of either the T_j predecessor of x_i or the T_j successor of x_i. Suppose that the search stops after tree T_f, having spent $O(2^f)$ time. It can be shown that $2^f = O(\log r_i)$.

As in Historical Insertion Sort we next have to insert x_i into all trees, as well as delete the oldest element from each tree. The insertion of x_i into trees $T_1, T_2, \ldots, T_f$ can be taken care of during the searching process. Further, insertion into T_K takes constant amortized time once the search has terminated. In Historical Insertion Sort the insertions into trees $T_{f+1}, T_{f+2}, \ldots, T_{K-1}$ relied on the knowledge that the node for x_i should be inserted *adjacent* to the thread associated with the node found in T_f; we call this node the 'anchor point' for the search and denote it by x_p. Here this assumption no longer holds. However, all nodes between x_i and x_p in $T_{f+1}, T_{f+2}, \ldots, T_{K-1}$ can be pruned without affecting the search time of subsequent insertions. Once these nodes have been pruned the thread associated with x_p can be used for guiding the insertions. The intuitive reason why the intervening nodes can be pruned is that no subsequent search for a position between x_i and x_p will use any of them as anchor point, since they are so old that either x_i or x_p (or some even more recently inserted node) will be used. The pruning process requires $O(n)$ comparisons and $O(n \log\log n)$ time over a sequence of n operations. The deletions of the oldest

elements are performed in exactly the same way as in Historical Insertion Sort, requiring $O(n \log\log n)$ time and no comparisons over a sequence of n operations.

Theorem 14. *Regional Insertion Sort spends* $C_{RIS}(X) = O(n + \log Reg(X))$ *comparisons and* $T_{RIS}(X) = O(n \log\log n + \log Reg(X))$ *time to sort any sequence* X *of length* n.

Proof. As in the proof of Theorem 12, the preceding discussion gives that the number of comparisons spent is given by

$$O\left(\sum_{i=2}^{n} \log r_i\right) = O(n + \log Reg(X)).$$

The time taken is linear in the number of comparisons plus $O(n \log\log n)$. □

Now, by Lemma 4, Theorem 14, and the fact that $C_{Reg}(n, k) = \Omega(n + \log k)$, we have the following corollary.

Corollary 15. $C_{Reg}(n, k) = \Theta(n + \log k)$.

The time bound stated in Theorem 14 is in fact a little more pessimistic than it could be. The $\log\log n$ term stems from that *every* inserted element is assumed to contribute to increasing the number of trees in the hierarchy. However, due to the pruning this need not always be the case. For instance, if x_i is inserted immediately adjacent to x_{i-1} then x_{i-1} can be pruned, leaving the number of nodes in the data structure, and thus the number of trees, unchanged. Hence, when presented with a sorted sequence the hierarchy will never grow, but consist of a single tree that contains one node, and the sorting will take linear time altogether. More generally, the number of trees in the data structure will be (at most) doubly logarithmic in the number of insertions that do not take place immediately adjacent to the previous insertion point. The number of such insertions is bounded from above by the number of blocks within the input sequence. (See the definition of *Block* in Table 1.) We can thus strengthen the time bound of Regional insertion sort to

$$T_{RIS}(X) = O(n \log\log Block(X) + \log Reg(X)).$$

7. The partial order revisited

We insert the three measures introduced in the previous section into the partial order. We further show that the resulting partial order shows all relations among the measures.

7.1. Inserting Loc

Theorem 16. *Loc* ⊃ *Block.*

Proof. Recall the definition of *Block* from Table 1, and consider any sequence X of length n. For the first element of each block in X we have $d_i \leqslant n$. The second and subsequent elements within any block will have $d_i = 1$. Thus,

$$Loc(X) = \prod_{i=2}^{n} d_i = O(n^{Block(X)}).$$

Using the bounds on $C_{Loc}(n, k)$ and $C_{Block}(n, k)$ from Table 1 gives

$$\begin{aligned} C_{Loc}(|X|, Loc(X)) &= \Theta(|X| + \log Loc(X)) \\ &= O(|X| + Block(X)\log|X|) \\ &= O(|X| + Block(X)\log Block(X)) \\ &= O(C_{Block}(|X|, Block(X))). \end{aligned}$$

The third equality can be verified as follows. Since both expressions are at least linear in $|X|$, the only possibility for the first one to be asymptotically greater than the second one is if its second term is superlinear. But this does not hold unless we have $Block(X) = \Omega(|X|/\log|X|)$. This in turn implies $\log Block(X) = \Omega(\log|X|)$, that is, $\log|X| = O(\log Block(X))$, from which the claim follows. Definition 4 now gives *Loc* ⊇ *Block.*

Proving that the superiority is strict is accomplished by considering the sequence $X = \langle 1, 3, 5, \ldots, n-1, 2, 4, 6, \ldots, n\rangle$. We have that $Loc(X) = O(n2^{n/2})$ and $Block(X) = n$. Since $C_{Loc}(n, n2^{n/2}) = \Theta(n)$ and $C_{Block}(n, n) = \Theta(n\log n)$, we have *Block* ⊉ *Loc* by Lemma 6. The result follows from Definition 4. □

Theorem 17. *Loc* ⊃ *Osc.*

Proof. Recall the definition of *Osc* from Table 1. We have

$$\begin{aligned} d_i &= \|\{j \mid 1 \leqslant j < i \text{ and } \min\{x_{i-1}, x_i\} < x_j < \max\{x_{i-1}, x_i\}\}\| + 1 \\ &\leqslant \|\{j \mid 1 \leqslant j \leqslant n \text{ and } \min\{x_{i-1}, x_i\} < x_j < \max\{x_{i-1}, x_i\}\}\| + 1, \end{aligned}$$

where the last expression is the same as the ith term in the sum defining *Osc*. Hence, for any sequence X of length n, $\sum_{i=2}^{n} d_i \leqslant n + Osc(X)$. The inequality relating the geometric and arithmetic means implies

$$Loc(X) = \prod_{i=2}^{n} d_i = O\left(\left(1 + \frac{Osc(X)}{n}\right)^n\right).$$

Combining this bound with those on $C_{Loc}(n, k)$ and $C_{Osc}(n, k)$ now yields

$$\begin{aligned} C_{Loc}(|X|, Loc(X)) &= \Theta(|X| + \log Loc(X)) \\ &= O\left(|X| + |X| \log\left(1 + \frac{Osc(X)}{|X|}\right)\right) \\ &= O(C_{Osc}(|X|, Osc(X))). \end{aligned}$$

It follows that $Loc \supseteq Osc$ by Definition 4.

To show that *Osc* and *Loc* are not equivalent, consider the sequence

$$X = \left\langle n, 1, n-1, 2, \ldots, \frac{n}{2}+1, \frac{n}{2} \right\rangle.$$

Here $Osc(X) = \Theta(n^2)$ and $Loc(X) = 1$. Now, as $C_{Osc}(n, n^2) = \Theta(n \log n)$ and $C_{Loc}(n, 1) = \Theta(n)$, Lemma 6 gives $Osc \not\supseteq Loc$, which completes the proof. □

7.2. Inserting Hist

The next two theorems show that *Hist* is a significant new measure of presortedness.

Theorem 18. *Hist* $\supset$ *Block*.

Proof. If we replace d_i by t_i and *Loc* by *Hist*, the same argument as in the proof of $Loc \supseteq Block$ in Theorem 16 gives $Hist \supseteq Block$.

To prove that the superiority is strict consider the perfect shuffle

$$X = \left\langle 1, \frac{n}{2}+1, 2, \frac{n}{2}+2, \ldots, \frac{n}{2}, n \right\rangle,$$

for which $Hist(X) = \Theta(2^n)$ and $Block(X) = n$. Since $C_{Hist}(n, 2^n) = \Theta(n)$ and $C_{Block}(n, n) = \Theta(n \log n)$, we have that $Block \not\supseteq Hist$ by Lemma 6. The result follows. □

Theorem 19. *Hist* $\supset$ *Inv*.

Proof. We first show that, for any sequence X of length n,

$$Hist(X) = O\left(\left(1 + \frac{Inv(X)}{n}\right)^n\right). \tag{2}$$

As was observed in the proof of Theorem 17, the bound follows if we show that $\sum_{i=2}^{n} t_i = O(n + Inv(X))$.

We consider the number of inversions induced *on* x_i and the number of inversions induced *by* x_i, $inv_l(x_i)$ and $inv_r(x_i)$, respectively:

$$inv_l(x_i) = \|\{j \mid 1 \leqslant j < i \text{ and } x_j > x_i\}\|,$$

$$inv_r(x_i) = \|\{j \mid i < j \leqslant n \text{ and } x_j < x_i\}\|.$$

Then $Inv(X) = \sum_{i=1}^{n} inv_l(x_i) = \sum_{i=1}^{n} inv_r(x_i)$. Consider the position of x_i, shown in Fig. 3, where $inv_l(x_i)$ is the number of elements in the upper left quadrant centered on x_i and $inv_r(x_{i-t_i})$ is the number of elements in the lower right quadrant centered on x_{i-t_i}.

By definition, t_i equals one plus the number of elements strictly included in the slab defined by vertical lines through x_i and x_{i-t_i}. For the case when $x_i > x_{i-t_i}$, which is what is shown in Fig. 3, we have

$$t_i \leqslant inv_l(x_i) + inv_r(x_{i-t_i}) + 1,$$

where the additive one is due to the fact that element x_{i-t_i} itself does not contribute to either of the preceding terms. The same relationship is easily seen to hold for the case when $x_i > x_{i-t_i}$.

Hence,

$$\begin{aligned}\sum_{i=2}^{n} t_i &\leqslant n + \sum_{i=1}^{n} inv_l(x_i) + \sum_{i=1}^{n} inv_r(x_{i-t_i}) \\ &\leqslant n + 3\sum_{i=1}^{n} inv_r(x_i) \\ &= O(n + Inv(X)),\end{aligned}$$

with the second inequality following from the observation that each x_{i-t_i} can be the most recent (left) neighbor for at most two other elements, one above and one below. We have thus verified that Eq. (2) holds.

$Hist \supseteq Inv$ now follows from Eq. (2) and the bounds on $C_{Hist}(n, k)$ and $C_{Inv}(n, k)$ by the same argument as was used to prove $Loc \supseteq Osc$ in Theorem 17.

To prove that the superiority is strict consider again the perfect shuffle used in the proof of Theorem 18. For such sequences X, we have $Hist(X) = \Theta(2^n)$ and

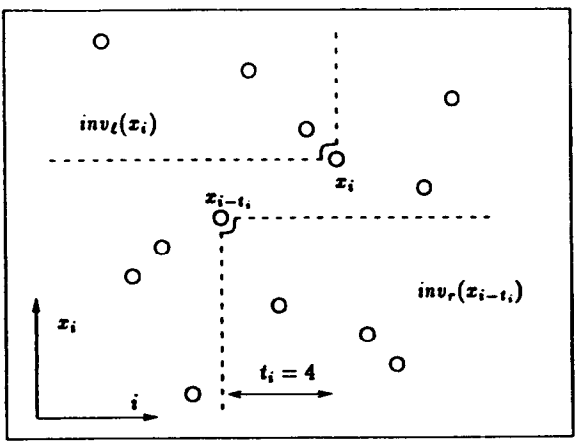

Fig. 3. Relationship between *Hist* and *Inv*.

$Inv(X) = \Theta(n^2)$. Since $C_{Hist}(n, 2^n) = \Theta(n)$ and $C_{Inv}(n, n^2) = \Theta(n \log n)$ the result follows by Lemma 6. □

7.3. Inserting Reg

Since *Reg* is a generalization of *Loc* and *Hist*, the following two theorems should be expected.

Theorem 20. *Reg* ⊃ *Loc*.

Proof. In the previous section we observed that $r_i \leqslant d_i$, and thus,

$$Reg(X) = \prod_{i=2}^{n} r_i \leqslant \prod_{i=2}^{n} d_i = Loc(X),$$

for any sequence X of length n. Hence,

$$\begin{aligned} C_{Reg}(|X|, Reg(X)) &= \Theta(|X| + \log Reg(X)) \\ &= O(|X| + \log Loc(X)) \\ &= O(C_{Loc}(|X|, Loc(X))). \end{aligned}$$

It follows that $Reg \supseteq Loc$ by Definition 4.

The relation is seen to be strict by once again considering the perfect shuffle from the proof of Theorem 18, for which $Loc(X) = \Omega((n/4)^{n/2})$, since $d_i \geqslant n/4$ for $n/2 < i \leqslant n$, while $Reg(X) = \Theta(2^n)$. The result follows from Lemma 6, since $C_{Loc}(n, (n/4)^{n/2}) = \Theta(n \log n)$ and $C_{Reg}(n, 2^n) = \Theta(n)$. □

Theorem 21. *Reg* ⊃ *Hist*.

Proof. Using the observation that $r_i \leqslant t_i$, the same argument as in the previous proof gives $Reg \supseteq Hist$.

To prove $Hist \not\supseteq Reg$, consider the sequence $X = \langle 1, 3, 5, \ldots, n-1, 2, 4, 6, \ldots, n \rangle$. Here $Reg(X) = \Theta((n/2)2^{n/2})$ and $Hist(X) = \Theta((n/2)^{n/2})$. Therefore, $Hist \not\supseteq Reg$, since $C_{Reg}(n, (n/2)2^{n/2}) = \Theta(n)$ and $C_{Hist}(n, (n/2)^{n/2}) = \Theta(n \log n)$. □

The next theorem might be more surprising than the preceding ones.

Theorem 22. *Reg* ⊃ *SMS*.

Proof. Recall the definition of the measure *SMS* from Table 1. Let X be a sequence of length n with $SMS(X) = k$. We show that $Reg(X) = k^{O(n)}$, from which it follows that

$$\begin{aligned} C_{Reg}(|X|, Reg(X)) &= O(|X| \log SMS(X)) \\ &= O(C_{SMS}(|X|, SMS(X))), \end{aligned}$$

proving $Reg \supseteq SMS$, by Definition 4.

Let X_j be any monotone subsequence in a decomposition of X into k monotone subsequences. Consider the sum of the r_i's taken over all elements in X_j:

$$\sum_{x_i \in X_j} r_i = \sum_{x_i \in X_j} \min_{1 \leqslant t < i} \{t + d_{i,i-t} - 1\} \leqslant \sum_{x_i \in X_j} t(i) + d_{i,i-t(i)},$$

where $t(i)$ may take any value $1 \leqslant t(i) < i$. We bound the sum from above by picking suitable values of $t(i)$. If x_i is the first element in X_j, let $t(i) = i - 1$. Otherwise, suppose x_l is the element preceding x_i in X_j, in which case we let $t(i) = i - l$. Then the sum of the $t(i)$'s will telescope, and totals to at most n. Similarly, the sum of the $d_{i,i-t(i)}$'s cannot exceed $2n$, since each of the elements *not* in X_j can be in at most one of the partitions created by the elements in X_j, and the first element in X_j contributes at most an additional n to the sum. We conclude that $\sum_{x_i \in X_j} r_i \leqslant 3n$. As this applies for every monotone subsequence, we have $\sum_{i=2}^{n} r_i \leqslant 3kn$. Since $Reg(X) = \prod_{i=2}^{n} r_i$ is maximized when the r_i's are about the same, we have

$$Reg(X) \leqslant \prod_{i=2}^{n} 3k = k^{O(n)},$$

which proves the first part of the claim, as stated above.

To show that the superiority is strict, consider the sequence X obtained from the sorted sequence by permuting the last $\sqrt{n}$ elements, so that SMS is maximized for that part of the sequence. Then $SMS(X) = \Theta(n^{1/4})$, while $Reg(X) \leqslant (\sqrt{n})^{\sqrt{n}}$. Since $C_{Reg}(n, (\sqrt{n})^{\sqrt{n}}) = \Theta(n)$ and $C_{SMS}(n, n^{1/4}) = \Theta(n \log n)$, Lemma 6 implies that $SMS \not\supseteq Reg$, which concludes the proof. □

Fig. 4 shows the partial order resulting from the theorems proved above.

7.4. *No relations are missing*

We show the following theorem.

Theorem 23. *Fig. 4 shows all relations among the measures.*

The theorem follows if we can verify that no arcs are missing in Fig. 4. In order to do this we must prove that several pairs of measures are independent. We first state a lemma that explicitly explains how the transitivity of $\supseteq$ will be used in this process.

Lemma 24. *For any measures M_1, M_2, M_3, and M_4 in $\mathcal{M}$, if $M_1 \supseteq M_2$, $M_3 \supseteq M_4$, and $M_1 \not\supseteq M_4$, then $M_2 \not\supseteq M_3$.*

Proof. Let M_1, M_2, M_3, and M_4 satisfy the assumptions, and assume that $M_2 \supseteq M_3$. Then, by transitivity, $M_1 \supseteq M_2 \supseteq M_3 \supseteq M_4$, which contradicts the assumptions and concludes the proof. □

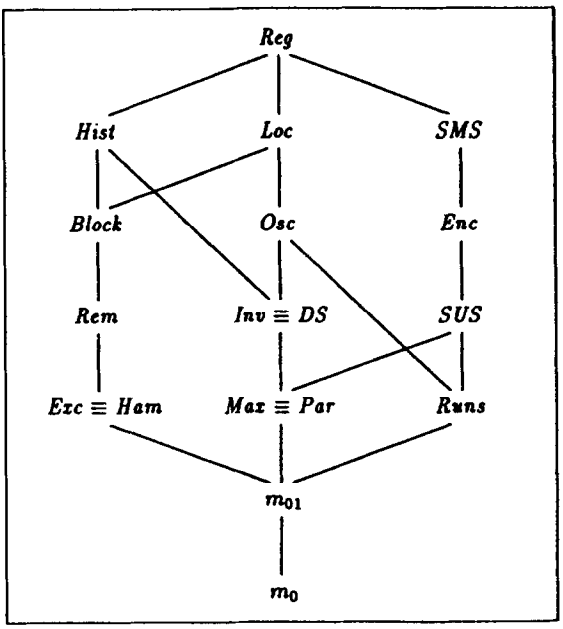

Fig. 4. $\mathcal{P}(\mathcal{M}, \supseteq)$ after insertion of *Loc*, *Hist*, and *Reg*.

The lemma generalizes the observation made at the end of Section 4, where we claimed that since $Rem \supseteq Exc$ and $Rem \not\supseteq Runs$, it follows that $Exc \not\supseteq Runs$.

In combination with Lemma 24, the following lemma shows that, except of what is shown in Fig. 4, no measure in the left chain, that is, none of *Hist*, *Block*, *Rem*, and *Exc*, is superior to any of the other measures.

Lemma 25. *$Hist \not\supseteq Runs$ and $Block \not\supseteq Max$.*

Proof. Consider the sequence $X = \langle 1, 3, 5, \ldots, n-1, 2, 4, 6, \ldots, n \rangle$. We have that $Hist(X) = \Theta((n/2)^{n/2})$ and $Runs(X) = 2$. Since $C_{Hist}(n, (n/2)^{n/2}) = \Theta(n \log n)$ and $C_{Runs}(n, 2) = \Theta(n)$, it follows that $Hist \not\supseteq Runs$ by Lemma 6.

To verify the second claim consider the sequence $X = \langle 2, 1, 4, 3, \ldots, n, n-1 \rangle$. Here we have $Max(X) = 1$ and $Block(X) = n$. Since $C_{Max}(n, 1) = \Theta(n)$ and $C_{Block}(n, n) = \Theta(n \log n)$ the claim follows from Lemma 6. □

The next lemma shows that, except of what is shown in Fig. 4, no measure is superior to any measure in the left chain.

Lemma 26. *$SMS \not\supseteq Exc$, $Osc \not\supseteq Exc$, and $Loc \not\supseteq Hist$.*

Proof. Consider a sequence X, obtained from a sorted sequence by rearranging the last $\sqrt{n}$ elements so as to maximize the *SMS* measure for just that part of the

sequence. Then we have $SMS(X) = \Theta(n^{1/4})$ and $Exc(X) = O(\sqrt{n})$. Since $C_{SMS}(n, n^{1/4}) = \Theta(n \log n)$ and $C_{Exc}(n, \sqrt{n}) = \Theta(n)$, Lemma 6 implies $SMS \not\supseteq Exc$.

For the second claim, start with a sorted sequence of n elements and permute the last $n^{3/4}$ elements so that *Osc* is maximized for just that part of the sequence. The resulting sequence X has $Osc(X) = \Theta(n^{3/2})$ and $Exc(X) = O(n^{3/4})$. Since $C_{Osc}(n, n^{3/2}) = \Theta(n \log n)$ and $C_{Exc}(n, n^{3/4}) = \Theta(n)$ the claim follows from Lemma 6.

To prove the third claim consider once more the perfect shuffle

$$X = \left\langle 1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, \ldots, \frac{n}{2}, n \right\rangle.$$

We have already noted that $Loc(X) = \Omega((n/4)^{n/2})$, and it is easy to see that $Hist(X) = \Theta(2^n)$. Since $C_{Loc}(n, (n/4)^{n/2}) = \Theta(n \log n)$ and $C_{Hist}(n, 2^n) = \Theta(n)$ the claim follows from Lemma 6. □

It remains to prove that no relations are missing in the right part in $\mathscr{P}(\mathscr{M}, \supseteq)$. This is done in the following two lemmas.

Lemma 27. *$SMS \not\supseteq Inv$ and $Loc \not\supseteq SUS$.*

Proof. The first claim follows by considering the same sequence that was used to prove the first claim in the previous lemma; for that sequence $SMS(X) = \Theta(n^{1/4})$ and $Inv(X) = O(n)$. Since $C_{Inv}(n, n) = \Theta(n)$ and $C_{SMS}(n, n^{1/4}) = \Theta(n \log n)$ the claim follows from Lemma 6.

To prove the second claim consider again the shuffle

$$X = \left\langle 1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, \ldots, \frac{n}{2}, n \right\rangle.$$

Here $Loc(X) = \Omega((n/4)^{n/2})$ and $SUS(X) = 2$. As $C_{Loc}(n, (n/4)^{n/2}) = \Theta(n \log n)$ and $C_{SUS}(n, 2) = \Theta(n)$ the claim follows from Lemma 6. □

Lemma 28. *$Inv \not\supseteq Runs$ and $Runs \not\supseteq Max$.*

Proof. Consider the sequence

$$X = \left\langle \frac{n}{2} + 1, \frac{n}{2} + 2, \ldots, n, 1, 2, \ldots, \frac{n}{2} \right\rangle.$$

We have that $Inv(X) = \Theta(n^2)$ and $Runs(X) = 2$. Since $C_{Inv}(n, n^2) = \Theta(n \log n)$ and $C_{Runs}(n, 2) = \Theta(n)$ it follows that $Inv \not\supseteq Runs$ by Lemma 6.

For the second claim, consider the sequence $X = \langle 2, 1, 4, 3, \ldots, n, n-1 \rangle$. We have that $Runs(X) = n/2$ and $Max(X) = 1$. Since $C_{Runs}(n, n/2) = \Theta(n \log n)$ and $C_{Max}(n, 1) = \Theta(n)$ it follows that $Runs \not\supseteq Max$ by Lemma 6. □

Theorem 23 now follows from Lemmas 24–28.

8. Conclusions

The partial order on measures of presortedness is important for two reasons. First, the formalism we have introduced for comparing measures of presortedness and the partial order established by this formalism mean that any proposed new measure can have its usefulness evaluated using the partial order as a yardstick. Moreover, since every sorting algorithm defines a curve crossing the partial order, the order can also be used for evaluating the usefulness of any adaptive sorting algorithm. The transitivity of the relation $\supseteq$ helps this process, since it is not necessary for the adaptivity of the algorithm to be investigated for every measure.

At the top of the partial order we have described three new measures, *Loc*, *Hist*, and *Reg*. Local Insertion Sort is *Loc*-optimal, and we have sketched algorithms adapting to the other two measures. The efficient implementation of Historical and Regional Insertion Sort remains an open problem. Should it exist, a *Reg*-optimal implementation of Regional Insertion Sort would be optimal for all known measures of presortedness.

Acknowledgements

We would like to thank Christos Levcopoulos, Svante Carlsson, Jyrki Katajainen, Heikki Mannila, and Arne Andersson for stimulating discussions. This work was in part supported by The Australian Research Council and the Swedish Board for Technical Development.

References

[1] A. Brandstädt and D. Kratsch, On partitions of permutations into increasing and decreasing subsequences, J. Inform. Process. Cybernet. 22 (1986) 263–273.
[2] W.H. Burge, Sorting, trees, and measures of order, Inform. and Control 1 (1958) 181–197.
[3] S. Carlsson, C. Levcopoulos and O. Petersson, Sublinear merging and Natural Mergesort, Algorithmica 9 (1993) 629–648.
[4] C.R. Cook and D.J. Kim, Best sorting algorithms for nearly sorted lists, Comm. ACM 23 (1980) 620–624.
[5] P. Diaconis and R.L. Graham, Spearman's footrule as a measure of disarray, J. Roy. Statist. Soc. Ser. B 39 (1977) 262–268.
[6] E.W. Dijkstra, Smoothsort, an alternative to sorting in situ, Sci. Comput. Programming 1 (1982) 223–233.
[7] G.R. Dromey, Exploiting partial order with Quicksort, Software Practice and Experience 14 (1984) 509–518.
[8] V. Estivill-Castro and D. Wood, A new measure of presortedness, Inform. and Comput. 83 (1989) 111–119.
[9] M.L. Fredman, How good is the information theory bound in sorting?, Theoret. Comput. Sci. 1 (1976) 355–361.

[10] L.J. Guibas, E.M. McCreight, M.F. Plass and J.R. Roberts, A new representation of linear lists, in: Proceedings of the 9th Annual ACM Symposium on Theory of Computing (1977) 49–60.
[11] J. Katajainen, C. Levcopoulos and O. Petersson, Local Insertion Sort revisited, in: Proceedings of the International Symposium on Optimal Algorithms, Lecture Notes in Computer Science 401 (Springer, Berlin, 1989) 239–253.
[12] D.E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching (Addison-Wesley, Reading, MA, 1973).
[13] C. Levcopoulos and O. Petersson, Adaptive Heapsort, J. Algorithms 14 (1993) 395–413.
[14] C. Levcopoulos and O. Petersson, Sorting shuffled monotone sequences, Inform. and Comput. 112 (1994) 37–50.
[15] C. Levcopoulos and O. Petersson, Splitsort–an adaptive sorting algorithm, Inform. Process. Lett. 39 (1991) 205–211.
[16] H. Mannila, Instance complexity for sorting and NP-complete problems, Ph.D. Thesis, Department of Computer Science, University of Helsinki, Helsinki (1985).
[17] H. Mannila, Measures of presortedness and optimal sorting algorithms, IEEE Trans. Comput. 34 (1985) 318–325.
[18] K. Mehlhorn, Sorting presorted files, in: Proceedings of the 4th GI Conference on Theoretical Computer Science 67 (Springer, Berlin, 1979) 199–212.
[19] K. Mehlhorn, Data Structures and Algorithms, Vol. 1: Sorting and Searching (Springer, Berlin, 1984).
[20] A. Moffat and O. Petersson, Historical searching, Internat. J. Found. Comput. Sci. 4 (1993) 85–98.
[21] O. Petersson, Adaptive sorting, Ph.D. Thesis, Department of Computer Science, Lund University, Lund (1990).
[22] S.S. Skiena, Encroaching lists as a measure of presortedness, BIT 28 (1988) 775–784.